Documentation of the demonstration that transfers user movements from the Kinect sensor to the Acroban robot

Damien Martin-Guillerez

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

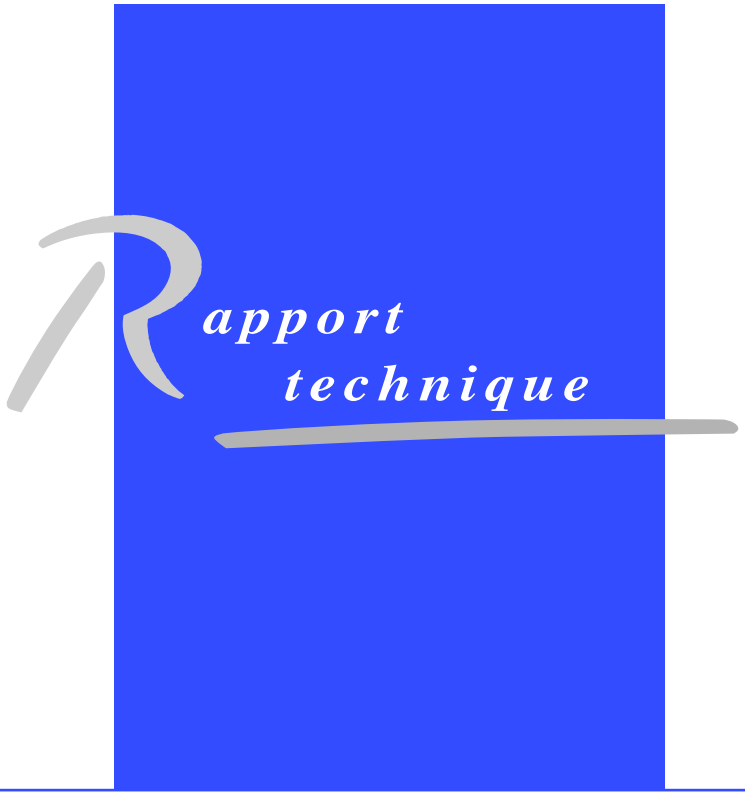
Documentation of the demonstration that transfers user movements from the Kinect sensor to the Acroban robot

Damien Martin-Guillerez

N° 0413

October 2011

Perception, Cognition, Interaction



ISSN 0249-0803 ISRN INRIA/RT--0413--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
BORDEAUX - SUD OUEST

Documentation of the demonstration that transfers user movements from the Kinect sensor to the Acroban robot

Damien Martin-Guillerez*

Domain : Perception, Cognition, Interaction
Équipe-Projet FLOWERS

Rapport technique n° 0413 — October 2011 — 16 pages

Abstract: This document describes a demonstration that uses a Kinect sensor† to copy the position of a user to the Acroban Robot‡. The Kinect sensor is a sensor able to build a 3-D representation of the user. It was developed by Microsoft for its video game system XBox 360§. The Acroban Robot is a lightweight robot used for experimenting on human-robot interface and robot learning inside the FLOWERS team. The demonstration described by this document is able to reproduce in real-time the movement of the user on the robot. This document describe how to run the demonstration, how to adapt it to different versions of the robot and how it works internally.

Key-words: Acroban, Kinect, inverse kinematic, demonstration

* Inria SED Bordeaux – Sud Ouest
† `http://www.xbox.com/kinect`
‡ `http://flowers.inria.fr/acroban.php`
§ `http://www.xbox.com`

Centre de recherche INRIA Bordeaux – Sud Ouest
Domaine Universitaire - 351, cours de la Libération 33405 Talence Cedex
Téléphone : +33 5 40 00 69 00

Documentation du démonstrateur copiant les mouvements de l'utilisateur sur le robot Acroban avec un capteur Kinect

Résumé : Ce document décrit une démonstration utilisant le capteur Kinect[¶] pour recopier la position de l'utilisateur sur le robot Acroban[‖]. Le capteur Kinect est un capteur capable de reconstruire une représentation 3-D de l'utilisateur. Il a été développé par Microsoft pour son système de jeux vidéo XBox 360[**]. Le robot Acroban est un robot léger utilisé pour expérimenter sur les interfaces homme-robot et l'apprentissage des robots dans l'équipe-projet FLOWERS. La démonstration décrite par ce document est capable de reproduire en temps réel les movements de l'utilisateur sur le robot. Ce document décrit comment lancer la démonstration, comment l'adapter aux différentes versions du robot et comment elle fonctionne.

Mots-clés : Acroban, Kinect, cinématique inverse, démonstration

[¶] http://www.xbox.com/kinect
[‖] http://flowers.inria.fr/acroban.php
[**] http://www.xbox.com

1 Introduction

This document describes a demonstration that uses a Kinect sensor[1] to copy the position of a user to the Acroban Robot[2]. The Kinect sensor is a sensor able to build a 3-D representation of the user. It was developed by Microsoft for its video game system XBox 360[3]. The Acroban Robot is a lightweight robot used for experimenting on human-robot interface and robot learning inside the FLOWERS team. The demonstration described by this document is able to reproduce in real-time the movement of the user on the robot.

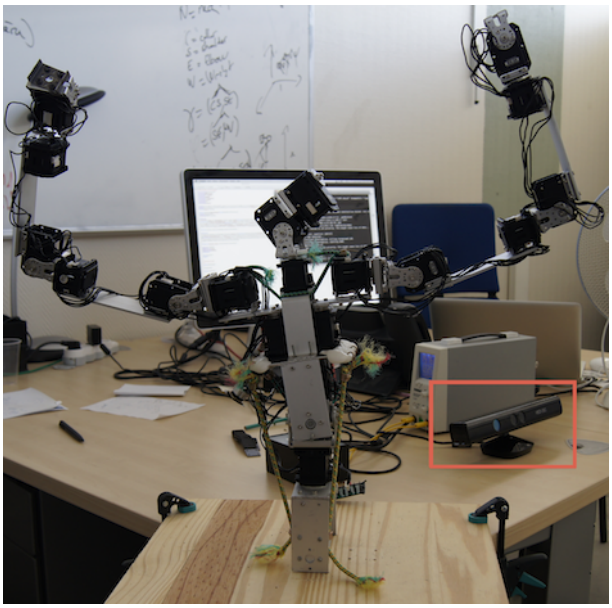


Figure 1: The Acroban torso and the Kinect sensor

The Kinect sensor is rounded with a red box

This demonstration was actually designed to work with the torso version of the Acroban robot shown in Figure 1. We can also see in that figure the Kinect sensor used to fetch the movements of the user. Of course this demonstration is easily extendable to other versions of the robot. The reader who wants to go through this document should first have a look at the Rhoban software used to control motors of Acroban and how to wire Acroban. The reader should also have a look at the OpenNI[4] software and install it on the demonstration computer. Finally, the reader wanting to read that document should be at ease using the Urbi framework[5] and have Urbi installed on his computer.

In this document, we will go into how to set-up, run and tune the demonstration in Section 2 and how the demonstration internals works in Section 3.

[1] `http://www.xbox.com/kinect`
[2] `http://flowers.inria.fr/acroban.php`
[3] `http://www.xbox.com`
[4] `http://openni.org`
[5] `http://www.urbiforge.org`

A short conclusion in Section 4 gives some pointers for the experimented user who want to go in the details of the demonstration code.

2 Running the demonstration

This section describes how to set-up the hardware of the demonstration in Section 2.1 and how to set-up the software of the demonstration in Section 2.2. Section 2.3 explains how the system calibrates for the robot and then how the demonstration unfold in Section 2.4. Section 2.5 explains the adaptation needed to work with other versions of the Acroban robot (or other robot based on the same motors).

2.1 Hardware set-up

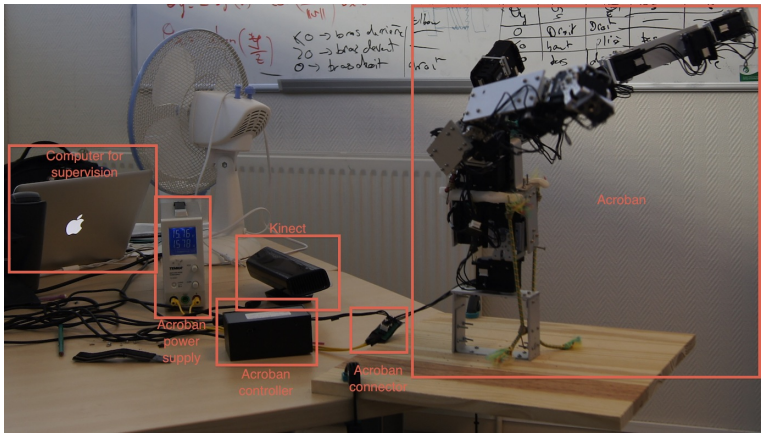


Figure 2: The hardware needed to run the demonstration

The system hardware is composed of three main components shown in Figure 2:

1. Acroban is the robot who has a specific power-supply and its own controller. The controller and the power supply is wired to Acroban through its connector. The controller runs a specific software developed by the FLOWERS team and this software can be controlled through a network service. The hardware of the controller also comes from the FLOWERS team.

2. The Kinect is the sensor that comes along with its own cables and power supply.

3. The computer supervises the demonstration. It is connected to the Acroban controller through a local ethernet network and to the Kinect through USB.

2.2 Software set-up

The software needed for the demonstration are:

- The Urbi framework available at `http://www.gostai.com/downloads/urbi/` is the framework used to orchestrate the system. The development of the demonstration was done was done using the 2.7.1 version under MacOSX (self-compiled, using a handmade patch to support the latest version of libboost[6]).
- OpenNI, SensorKinect and the Nite MiddleWare are libraries used to grab content from the Kinect device. A full explanation on how to install those libraries can be found on the team wiki[7].
- UObjects are Urbi primitives to talk to the hardware or analyze data. Four UObjects, that can be found in the uobjects directory of the gforge project Flowers Urbi Modules[8], are used in that demonstration:
  - UKinect is a UObject that is able, using OpenNI, to grab the skeleton of users in front of the Kinect and pass it to Urbi through a list of joints,
  - TrunkWindow is a UObject that renders a skeleton in a window,
  - AngleCalculator is a UObject that can convert positions of joints to rotation angles between bones, and
  - URhoban is the UObject to control the rhoban software, it requires the rhoban library.
- There are four scripts, that can be found in the projects/AcrobanKinect directory of the Flowers Urbi Modules project, to orchestrate the demonstration:
  - *utils.u* contains general purposes functions for the urbiscript part of the demonstration,
  - *acroban.u* contains the functions and classes to control the robot,
  - *demo.u* contains the demonstration set-up, and
  - *run.u* is a launcher that calls the demonstration function.
- A launcher, *run.sh*, is provided to run the demonstration.
- A configuration file, *config.ini*, enables tuning the demonstration for the hardware. In particular, it has `host` and `port` entries in the `general` section that specify where the rhoban server can be found.

To run the demonstration, one should make sure to have all the software installed and to have the UObjects in a modules subdirectory of the scripts directory. The computer should be connected to the Kinect and the Acroban controller. The network configuration should make the controller accessible by the supervised computer and the IP address as well as the TCP port of the

[6]The page `https://wiki.bordeaux.inria.fr/flowers/doku.php?id=soft:urbi:compile` explains how to build Urbi 2.7.1 for MacOSX

[7]`https://wiki.bordeaux.inria.fr/flowers/doku.php?id=soft:devices:kinect`

[8]`https://gforge.inria.fr/projects/fum/`

rhoban server correctly entered in the configuration file. The demonstration is simply ran by launching `./run.sh` in a terminal. One can add the `-r` option to the command to ignore the robot and run the demo without the robot.

2.3 Calibration

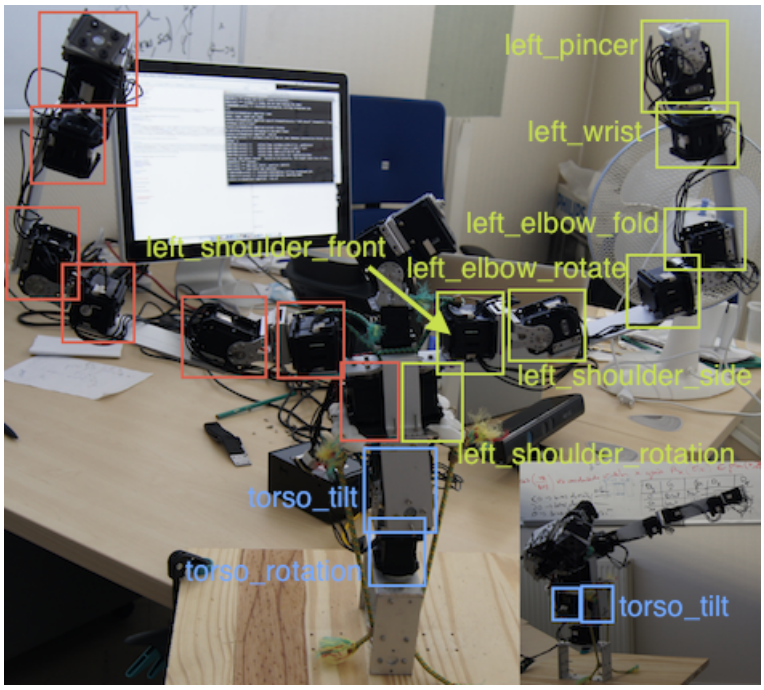


Figure 3: The motors name for the Acroban trunk

Right names are the same as left names but with the prefix right instead of left

As said before, to run the demonstration, one should launch it with the `./run.sh` command in a terminal. This starts the whole demonstration. Make sure the robot is correctly powered (i.e., the power supply is turned on, the embedded controller is wired and that the power supply output is turned on).

When the demonstration is launched, it looks for a *calibration.txt* file in the script directory. This file contains the calibration of each motor of the robot (i.e., for each motor name, its identifier and the several possible positions). Figure 3 shows the names used for the motors of the Acroban trunk. If the *calibration.txt* file is not found or does not contain sufficient information, a calibration step is launched. It tries to identify each motor and the required motor positions.

For each motor it has to identify, the program will print "`Trying to identify motor XXX, please move it many times`" then the user can move the robot motors that are not identified. Move several time the motors until the robot block. The program will then display that it has identified the motor ("`Identified motor YY as motor XXX, can you move it? (set question to 'y' or 'n')`") and put only the identified motor in compliant mode (i.e., only this motor can

be moved). If the motor can be moved, type `y;↩` in the console and it will go to next step (and print "`Identified motor YY (key = ZZ) as motor XXX !`"). If it is not the good motor, answer `n;↩` and the program will block this motor and ask you to move the good motor again. If no more motor are available, the system will either fail if the motor is mandatory for the demonstration or ignore the motor.

Once a motor is found, the system calibrates the different motor position needed to calibrate the motor. So after announcing the calibration ("`Calibrating motor XXX (id YY)`"), it will ask the user to move the motor into a certain position ("`Please put motor XXX in position POS`") and the user can move the motor. Every time the system found a stable value it prints "`Found value N, does that seems correct?`" and the user has to answer by `y;↩` (in which case the system goes to the next value or the next motor) or by `n;↩` (in which case the system will reread for the motor value).

This is repeated for every motors and every values. After the calibration, it is saved into the file *calibration.txt*. There is a sample *calibration.txt* in the projects/AcrobanKinect directory of the Flowers Urbi Modules project (Listing 1). To force a recalibration, simply remove the file. You can also edit by hand this file. Its format is very simple: every line contains the motor name then its identifier and a series of couple (value name, value) separated by a colon (`<motorname> <id> [<name>:<value>]*`).

Listing 1: The *calibration.txt* file

```
right_shoulder_rotation 11 fixed:575
left_shoulder_rotation 12 fixed:889
left_pincer 5 closed:516
right_pincer 10 closed:344
left_wrist 4 natural:562
right_wrist 9 natural:924
torso_rotation 20 fixed:263
torso_tilt 13 fixed:660

right_shoulder_front 6 top:828 back:100 low:190
left_shoulder_front 16 top:52 back:950 low:671

left_shoulder_side 1 top:755 folded:197 straight:530
right_shoulder_side 7 top:718 folded:143 straight:450

left_elbow_fold 3 straight:500 folded:841
right_elbow_fold 15 straight:575 folded:907

right_elbow_rotate 8 straight:585 front:950 back:100
left_elbow_rotate 2 straight:882 back:950 front:100
```

2.4 Demonstration

When the user run the launcher, the demonstration starts by the robot going into free mode for calibration. If necessary the calibration is done as described in the previous section, then the robot goes into its start position with both arm towards top as shown in Figure 4. The system initializes the Kinect and displays "`Started`" when everything is ready.

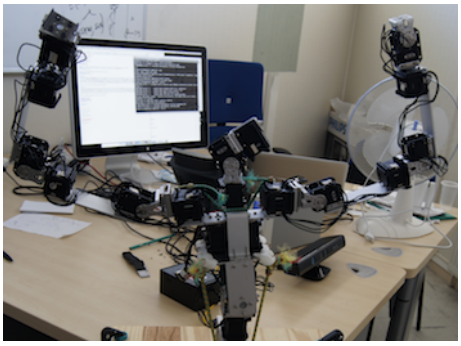


Figure 4: The start position of the robot, with both arms towards top

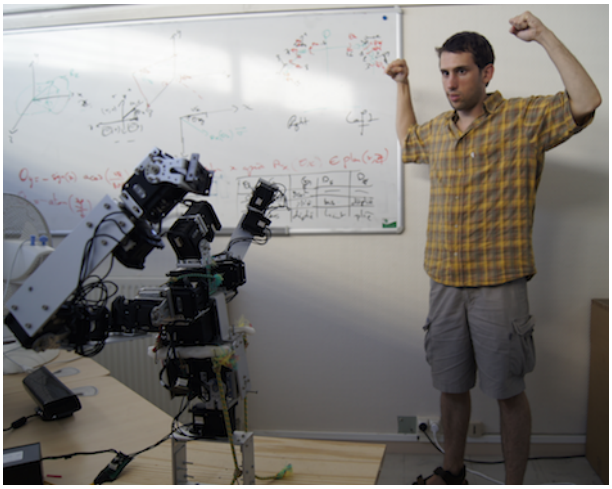


Figure 5: The start position of the user

The user then move in front of the Kinect. It then should be detected by the Kinect showing a "`New user...`" line on the terminal. The user can switch to the position with both arm towards top (Figure 5). It is the calibration position of the Kinect. When the Kinect starts calibrating it will display a "`Calibrating user`" line on the terminal. If this line is not displayed then the user has to adjust a bit its position. After a few seconds the user get calibrated and a message is displayed saying that the demo is running ("`Acroban is ready, we are now running the demo!`"). A window displaying the joint positions of the user appears. The robot then reproduces the position of the user arms (Figure 6).

The window showing the skeleton (Figure 7) have a main view that is a 2D projection of the 3D skeleton using any vector. It can be moved by moving

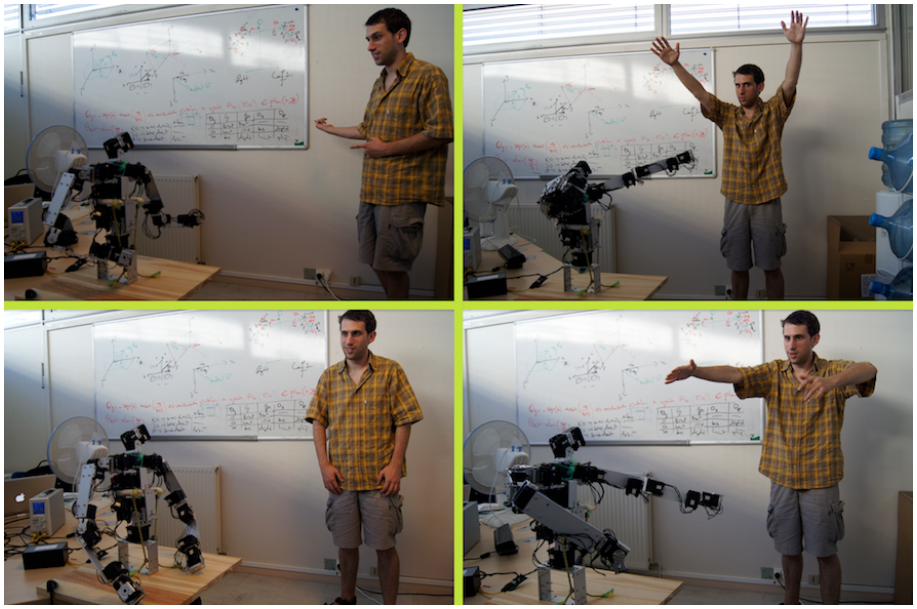


Figure 6: The robot arms are in the same position than the user's ones

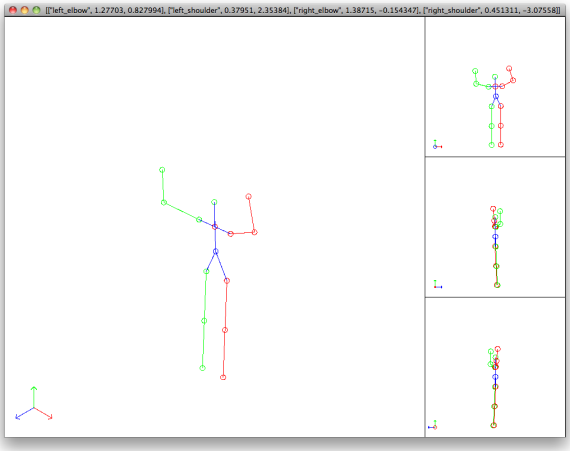


Figure 7: The window showing the skeleton

the mouse with the right button down, zoomed by using the mouse wheel and translated by moving the mouse with the left button down. The three other views are front projection, left and right projection. They can be zoomed and

translated but not rotated. The joints and bones whose name is prefixed by `left_` are colored in green, the one prefixed by `right_` are colored in red and the ones without left or right prefix are colored in blue. The 3-D axis given in the lower left of the views display the X axis in red (right), the Y axis in green (top) and the Z axis (front) in blue.

The demonstration continues to run until it is killed. If the user leaves the field of view of the Kinect and the Kinect lost it, the system will look for another user and stops the demo until a new user is found and calibrated. Some errors might appears in the reproduction of the position because: 1/ motor movements are limited and some movements cannot be reproduced, and 2/ the Kinect is based on learning methods and might sometimes guess wrong, especially when the user is standing with his back in front of the Kinect. Also the system might be unstable depending on the operating system because OpenNI has some flaws and random crash happens.

2.5 Adaptation to other robot

Some modifications of the *config.ini* file enable the system to adapt to any robot using the Bioloid [9] motors and the Rhoban server. The `general` section of this file contains the host and port to the server but also the radian per unit to convert between angles in radian and motor values (this value was taken directly from the RX28 Manual). This radian per unit can contains any Urbi expression. Thus the default `general` section is:

```
[general]
host=192.168.0.5
port=1234
radianPerUnit=(60 * Math.pi) / (36*1024.0)
```

Then a `motors` section provides the list of motors:

```
[motors]
torso_rotation = fixed :optional
torso_tilt = fixed :optional
left_shoulder_rotation = fixed :optional
right_shoulder_rotation = fixed :optional

left_shoulder_side = folded:max top:min straight:zero
right_shoulder_side = folded:max top:min straight:zero
left_shoulder_front = back:max low:zero top:min
right_shoulder_front = back:min low:zero top:max
left_elbow_rotate = straight:zero back:max front:min
right_elbow_rotate = straight:zero back:min front:max
left_elbow_fold = straight:zero:min folded:max
right_elbow_fold = straight:zero:min folded:max

left_wrist = natural :optional
right_wrist = natural :optional
left_pincer = closed :optional
right_pincer = closed :optional
```

Each motor that needs calibration has an entry in that section. The name of the motor is the entry key and the entry value gives the list of motor values

[9] `http://www.robotis.com/xe/bioloid_en`

to calibrate. Each motor value can be qualified as the value for angle zero by appending `:zero` after the value name, but also as the maximum (with the `:max` qualifier) or the minimum (`:min`) value that the motor is allowed to move to. Theses values will permits to convert between an angle in radian to a motor value. A last qualifier that might appears in the list of value names is the qualifier `:optional` to says that this motor is not needed for the demonstratoin.

Then a `start` section contains the angles to set at the start of the demonstration for each motor allowed to move:

```
[start]
left_shoulder_front = Math.pi
left_shoulder_side = Math.pi/6
left_elbow_fold = Math.pi/2
left_elbow_rotate = 0
right_shoulder_front = -Math.pi
right_shoulder_side = Math.pi/6
right_elbow_fold = Math.pi/2
right_elbow_rotate = 0
```

Each entry key is the motor name and each entry value is an angle in radian to set the start position of the motor.

Finally, the section `chains` gives the list of chains to convert from the skeleton to the robot:

```
[chains]
right_arm = -head neck \
    right_shoulder:right_shoulder_side:right_shoulder_front \
    right_elbow:right_elbow_fold:right_elbow_rotate right_hand
left_arm = -head neck \
    left_shoulder:left_shoulder_side:left_shoulder_front \
    left_elbow:left_elbow_fold:left_elbow_rotate left_hand
```

Each entry key is ignored, each entry value give the list of joints to use from the Kinect. Each joint can be qualified by two motors giving the rotations of the joint to copy. To fully understand these chains, the reading of the next section is needed.

3 Internals

This section presents how the demonstration works internally at the software level. Figure 8 gives the flowchart of the system. As presented in Section 3.1, the system grabs a list of joints from OpenNI using the UKinect UObject then transforms them into angles by passing those joints and a list of chains to the AngleCalculator UObject. This transformation is presented in Section 3.2. This list of joints is also passed to the TrunkWindow UObject for display. Finally, the angles are translated into motor values using the Acroban class in Urbi as presented in Section 3.3. This class transmits those values to the URhoban UObject to apply it to the robot.

3.1 Kinect output

Using the UKinect object, a list of joints along with their 3-d position is grabbed from the Kinect by the *demo.u* file. Joints that are provided by the objects are currently:

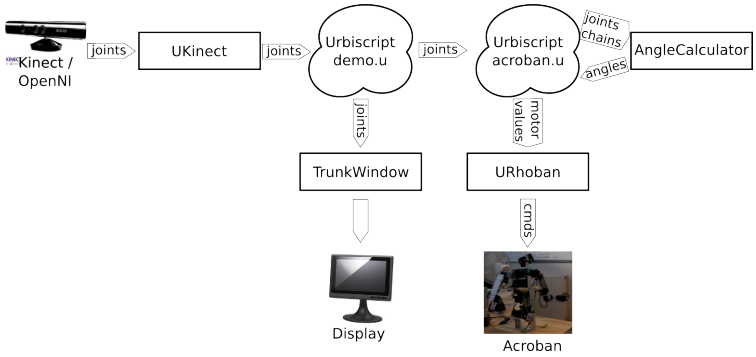


Figure 8: Flowchart of the software internals

- for the trunk, `neck`, `head`, `torso` and `waist` ;
- for the left arm, `left_collar`, `left_shoulder`, `left_elbow`, `left_wrist`, `left_hand`, `left_fingertip` ;
- for the right arm, `right_collar`, `right_shoulder`, `right_elbow`, `right_wrist`, `right_hand`, `right_fingertip` ;
- for the left leg, `left_hip`, `left_knee`, `left_ankle` and `left_foot` ;
- for the right leg, `right_hip`, `right_knee`, `right_ankle` and `right_foot`.

Joint names are pretty straightforward. Some joint positions are collapsed to the head position for unknown reason (this comes from OpenNI). In the current configuration of the demo only the head, neck, shoulders, elbows and hands are used.

The *demo.u* script provides a method `runKinectTest(refreshRate = 0.03s, w = 1000, backup = "test")` to simply run that part and display it on the TrunkWindow UObject. The `refreshRate` argument gives the period between each refresh of the window, the `w` argument gives the window's width and the `backup` argument gives in which file the measured skeleton positions will be saved. If the `backup` argument is nil then no backup will be performed. This allows to backup a set of positions to reload it later using the `loadTest(file)` function and then to replay it with the other functions of the *demo.u* file.

3.2 From the skeleton to arm angles

As stated in previous section, only the head, neck, shoulders, elbows and hands position are used in the current position. They define chains of joints (head, neck, shoulder, elbow, hand) giving us 4 segments for both chain (right and left). Figure 9 shows a skeleton with only the considered joints and segments.

To work out the rotations for each joint we want to, we first need to have a vector base. Indeed, the computation of oriented angles in 3D is intricate if based on cross and scalar products. Thus it is simpler and more robust to use a vector base and compute the Euler angles in that base.

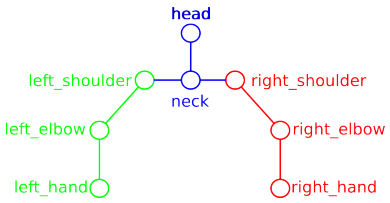


Figure 9: The used joints

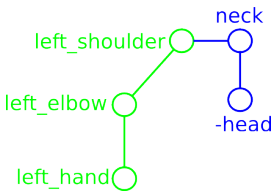


Figure 10: The left chain

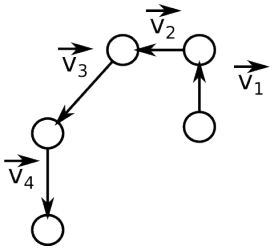


Figure 11: The four considered vectors in the left chain

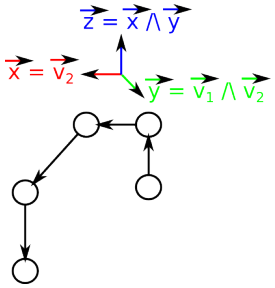


Figure 12: Starting base for the left chain

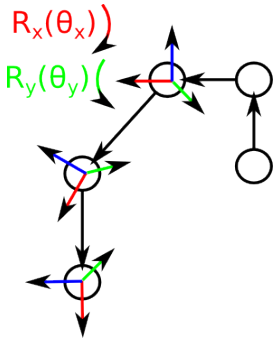


Figure 13: Rotations for the left chain

To get that rotation base, for convention reason, we have the minus operator (-) in the description of chains in the `chains` section of the configuration file. If we look at the joint list for the left arm from Section 2.5, we can observe that the actual list is `-head, neck, left_shoulder left_elbow left_hand`. When a minus sign is present, then the vector from this joint to the next is reverted. This gives us, from the chain of Figure 9, the chain of Figure 10. This

gives us four vectors that can be seen in Figure 11. As shown in Figure 12, we define the vector base using the two first vectors by[10]:

- $\overrightarrow{x} = \frac{\overrightarrow{v_2}}{||\overrightarrow{v_2}||}$
- $\overrightarrow{y} = \frac{\overrightarrow{v_1} \wedge \overrightarrow{v_2}}{||\overrightarrow{v_1} \wedge \overrightarrow{v_2}||}$
- $\overrightarrow{z} = \overrightarrow{x} \wedge \overrightarrow{y}$

Using that vector base, we can compute the angles of the next vector ($\overrightarrow{v_3}$) around the X and the Y axis as shown in Figure 13. We first compute the angle around the X axis as the rotation needed for the $\overrightarrow{v_3}$ vector to be in the (X,Z) plan. It is computed by $\theta_x = sign({v'_3}^y) \cdot cos^{-1}\left(-\frac{{v'_3}^z}{\sqrt{({v'_3}^y)^2+({v'_3}^z)^2}}\right)$[11] where $\overrightarrow{v'_3} = M \cdot \overrightarrow{v_3}$, M being the matrix to switch to our local base.

The angle around the Y axis is computed by first rotating the base according to θ_x: $M' = R_x(-\theta_x) \cdot M$. We obtain the angle of the rotation around the Y axis with the formula: $\theta_y = sign(-{v''_3}^z) cos^{-1}\left(\frac{{v''_3}^x}{|\overrightarrow{v''_3}|}\right)$ where $\overrightarrow{v''_3} = M' \cdot \overrightarrow{v_3}$.

Thanks to those both angles, we can compute $M'' = R_y(-\theta_y) \cdot R_x(-\theta_x) \cdot M$ the matrix of the base at the next point (see Figure 13). We can thus, by repeating those operations, computess the rotation θ_x and θ_y for each joints between the third and the penultimate in the chain.

This whole computation is done by the AngleCalculator UObject. It takes the description of the chain and the list of joint positions as input and returns (θ_y, θ_x) for each joints that is between the third and the penultimate in any chain.

3.3 Acroban representation

The *acroban.u* scripts provides two classes to handle the robot. The robot is seen as a series of motors. A motor is a UMotor provided by the URhoban UObject. It is encapsulated into the class CalibratedMotor that handles the calibration of the motor. This class provides the methods `applyAngle(angle, speed)` which takes an angle in radian, converts it to the motor value (control in position) according to its calibration (see Section 2.3) and applies it to the motor. The `speed` argument gives the maximum speed, in radian per second, to apply to the motor. The method `getAngle` do the opposite: it returns the current position, in radian, of the motor.

The second class to handle the robot is the Acroban class. It contains the list of motors discovered during the calibration phase. This class does the motor detection during the calibration using the UBioloid object provided by the URhoban UObject. This class also have save / restore functions for the calibration file. Finally it has functions to transfer the Kinect list of joints to the motors. Thus, the important functions of this class are:

- `backup(file)`, `restore(file)` and `identifyMotors()` does, respectively, the backup of the calibration, the restoration of the calibration, and the

[10] We use the $\cdot \wedge \cdot$ notation for the cross product and the $||\cdot||$ notation for the norm 2.
[11] v^x, v^y and v^z gives us the three coordinates of the vector $\overrightarrow{v}$

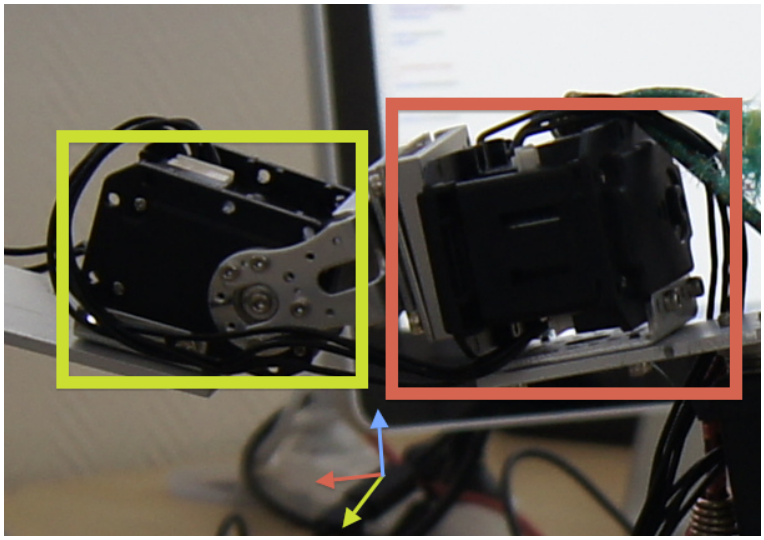


Figure 14: Example of two motors for a joint

The motor for the rotation around the Y axis (green axis) is surrounded by a green frame and the one around the X axis (red axis) is surrounded by a red frame

calibration itself. These three methods are called at the initialization of the object.

- `applyAngles(joint, angles, speed)` apply angles to the specific joint given by the argument `joint` which is a list of two motors name: the first for the rotation around the Y axis (θ_y, i.e., the folding of the arm) and the second for the rotation around the X axis (θ_x, i.e., the rotation of the arm). We can see in Figure 14 a joint with the two motors and the axis. The `angles` parameter is a list containing the two angles (θ_y, θ_x).
- `initTransfer(user)` prepare the robot for a transfer from a UKinect. The `user` parameter is a UKinectUser object from which the joint positions will be fetched.
- `addChain(joints)` adds a chain to transfer. This chain (the `joints` parameter) can be either a list or a string. It is of the same format that the one seen in Section 2.5. Each joints can be negated as seen in previous section, and each joint between the third one and the penultimate one can be completed with the name of two motors: the motor for the Y rotation and the motor for the X rotation. Therefore, the chain contained in the `chains` section of the *config.ini* file is of the form: "`<joint> <joint> (<joint>(:motorX:motorY)?)* <joint>`".
- `addChains()` adds the list of chains contained in the `chains` section of the *config.ini* file using the `addChain` function.
- `transfer(speed)` reads the list of joint positions from the UKinectUser passed to the `initTransfer` function, transfers them to the AngleCalcula-

tor UObject to compute the (θ_x, θ_y) for each joints between the third and the penultimate of each chains declared through `addChain`, and applies the computed angles to each motor declared in the chain.

To summarize, we can say that the robot is seen as a series of motors. The user is seen as a series of chains of joints. Each joint of a chain between the third and the penultimate can be mapped to two motors of the robot: one for the rotation around the Y axis (basically, this is the folding of an arm) and one for the rotation around the X axis (rotation around the arm).

Two methods from the *demo.u* scripts initialize the Acroban class and use the `transfer` function: `runAcrobanTest(test)` run the test set given in parameter and `runKinectAcroban` launch the whole demonstration (the *run.u* scripts simply launch that method).

4 Conclusion

We described how to run the demonstration, how to configure the demonstration and how the internal works.

This demonstration uses several components and thus requires other competencies. Software that have been specifically developed for this demonstration have Doxygen developer documentation, README and extensive code documentation. The interested reader should look into it. It can be found in the project AcrobanKinect (urbi scripts for handling the robot and setting up the demonstration), and in the UObjects UKinect (handling the kinect sensor), AngleCalculator (computing the angles according to section 3.2) and TrunkWindow (displaying the skeleton).

Centre de recherche INRIA Bordeaux – Sud Ouest
Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex (France)

Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803